An Introduction to Object-Oriented Programming and **Smalltalk**

Lewis J. Pinson

Richard S. Wiener

University of Colorado at Colorado Springs

ADDISON-WESLEY PUBLISHING COMPANY

Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam • Bonn
Sydney • Singapore • Tokyo • Madrid • San Juan

Sponsoring Editor: James T. DeWolf
Production Supervisor: Bette J. Aaronson
Production: Michael Bass & Associates
Manufacturing Supervisor: Hugh Crawford
Text Design: R. Kharibian & Associates
Cover Design: Marshall Henrichs

Library of Congress Cataloging-in-Publication Data

Pinson, Lewis J.
An introduction to object-oriented programming and Smalltalk / by Lewis J. Pinson and Richard S. Wiener.
p. cm.
Bibliography: p.
Includes index.
ISBN 020119127
1. Object-oriented programming (Computer science) 2. Smalltalk (Computer program language) I. Wiener, Richard, 1941– II. Title.
QA76.6.P56 1988
005.13'3—dc19

88-466
CIP

ABCDEFGHIJ-DO-898

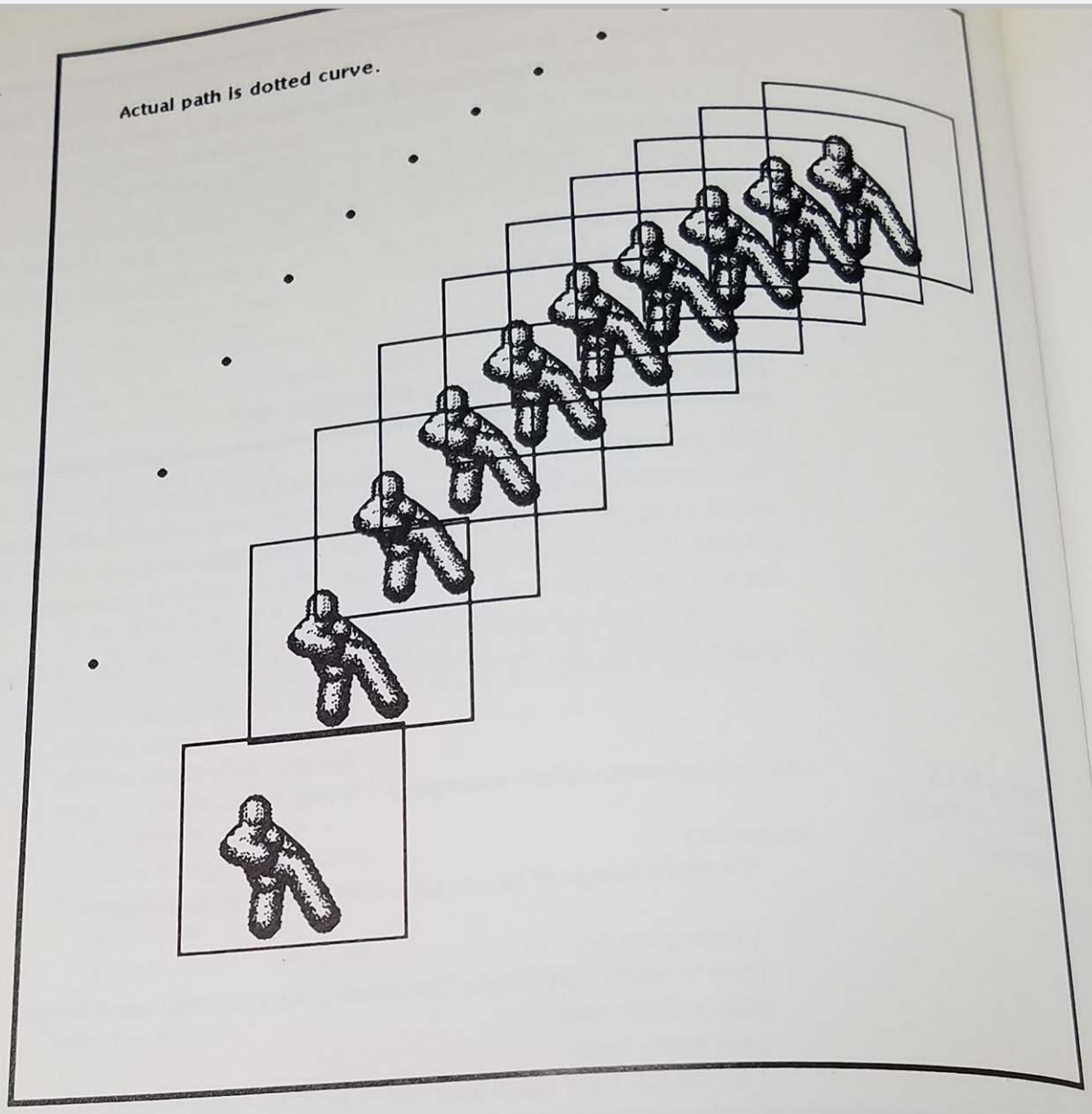

Figure 9.16 **Path** example1—Drawing with Complex Forms

Figure 9.16, the form has an offset of approximately 75@75 from the path point. This is a feature of the form as it is stored in file 'man4.form'. The display of the complex form is done using an OR rule (code 7).

9.6 The Model-View-Controller Triad

Any window on the Smalltalk screen has three essential components associated with it, the *model, view,* and *controller.* Together, these three components are

called the model-view-controller (MVC) triad. Any subview within a window can have an MVC triad that is different from its superview or other subviews. Some views are noninteractive, meaning they never have a need for acquiring control. For consistency with the MVC concept, these views have a controller that is an instance of class **NoController.**

The functions of the three components of the MVC triad and their properties are first described in Section 9.6.1. In Section 9.6.2, summaries are given for the abstract superclasses **View** and **Controller.** Finally, three examples are given that illustrate the MVC concept. The first example (in Section 9.6.3) uses an existing MVC triad in the Smalltalk system that is relatively simple. The second example (in Section 9.6.4) creates a new (and even simpler) MVC triad. A detailed discussion of what actually happens when this MVC application is invoked is given in Section 9.6.5. The third example illustrates the use of a valuable tool for investigating the properties of any MVC triad. Opening an inspector on a MVC triad is discussed in Section 9.6.6.

9.6.1 Interaction of model, view, and controller

The three components of the MVC triad have the following functionality and properties:

- *model*—an object that represents the data to be displayed. The model can be an instance of any appropriate class that represents the data most efficiently. For simplicity and elimination of extensive new protocol for all classes that can serve as models, the protocol of the model has no internal reference to its view or controller. Since the model contains information relative to the amount of data to be displayed in a view, it is often desirable to add a method for computing a bounding box to the model protocol.
- *view*—an object that is the visual display of the data represented by the model. Protocol in the view subclass determines how the data are to be displayed. Keeping the MVC triad together as a family with consistent information is the primary responsibility of the view. Private data within the view protocol are used to connect the view with its model and controller. A view consists of a top view and zero or more subviews. In many cases the top view is an instance of **StandardSystemView.** This has the advantage of providing the standard blue button menu options for collapsing, framing, moving, etc., for the new view. There are a number of key methods in **View** that need to be redefined in a new subclass of **View.** These include **displayView, defaultWindow,** and **defaultControllerClass.**
- *controller*—an object that provides the protocol for user interaction with the view and its model. Protocol in the controller subclass provides menus and action messages for the view and its model. Private data within the controller protocol are used to connect the controller with its model and view. The controller maintains an interface with the controller manager for coordination with other active controllers. In most cases control is

acquired through positioning of the cursor in combination with pressing or clicking the mouse buttons. Therefore, most controller classes are subclasses of **MouseMenuController,** which is a direct subclass of the superclass of all controllers, **Controller** (see Figure 9.1). Key methods in **Controller** that need redefinition in most subclasses include initialize-YellowButtonMenu, isControlActive, controlInitialize, and controlTerminate.

The basic sequence of events for the operation of MVC triads is given below. The steps outlined here are general in nature. More detailed explanations for the exact steps required to invoke an MVC application are given in Section 9.6.5.

1. New MVC applications are invoked with instance creation methods in the view class of the application. Sending one of these methods to the view class causes creation of the view, display of the model, and passage of control to the controller for the application. There are key methods in the view and controller class hierarchy that are part of this process. A more detailed discussion of these key methods is given in Section 9.6.5 for the histogram MVC example presented in Section 9.6.4.

2. When a new MVC triad is created, its controller is added to a list of scheduled controllers. This list is an instance variable, scheduledControllers, in class **ControlManager.** The global variable **ScheduledControllers** is the single instance of **ControlManager.** Protocol in **ControlManager** provides capability for accessing, scheduling, and modifying any controller in the list of scheduled controllers. This is accomplished by sending appropriate messages to **ScheduledControllers**

3. **ScheduledControllers,** the global instance of **ControlManager,** constantly monitors the position of the cursor and the state of the mouse buttons. When the cursor is moved within a view and a button is clicked, **ScheduledControllers** gives control to the controller for that view. The controller for the view is established as the value of another instance variable, activeController, in the control manager.

4. When a controller becomes the active controller, it receives the following sequence of messages from the startup message to activeController.

```
self controlInitialize.
self controlLoop.
self controlTerminate.
```

The active controller retains control and stays in the controlLoop until it relinquishes control. Of course, the user can move the cursor to another view and click a mouse button to terminate and change the active controller. Control can also be terminated by pressing certain special keys on the keyboard. The details of the controlLoop message are

```
controlLoop
        self isControlActive whileTrue: [
                Processor yield. self controlActivity].
```